```python
import numpy as np
import time
import matplotlib.pyplot as plt
```

```python
def rosenbrock(params):
    x, y = params
    return (1 - x)**2 + 100 * (y - x**2)**2

def grad_rosenbrock(params):
    x, y = params
    dx = -2 * (1 - x) - 400 * x * (y - x**2)
    dy = 200 * (y - x**2)
    return np.array([dx, dy])

def sine_reciprocal(x):
    # Handling x=0 by setting f(0)=0 and adding epsilon for gradient stability
    val = x[0] if isinstance(x, np.ndarray) else x
    if abs(val) < 1e-7: return 0.0
    return np.sin(1/val)

def grad_sine_reciprocal(x):
    val = x[0] if isinstance(x, np.ndarray) else x
    if abs(val) < 1e-7: return np.array([0.0])
    return np.array([-np.cos(1/val) / (val**2)])
```

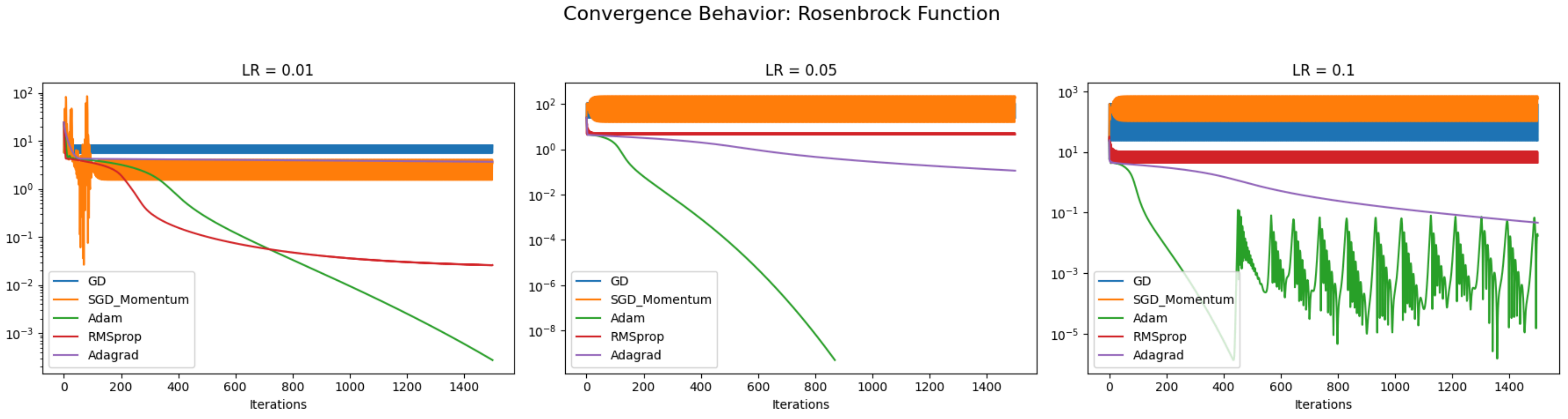
```python
# 2. Optimizer Implementations
class ScratchOptimizers:
    def __init__(self, lr):
        self.lr = lr
        self.m = 0
        self.v = 0
        self.t = 0

    def apply(self, name, w, grad):
        self.t += 1
        if name == 'GD':
            return w - self.lr * grad
        elif name == 'SGD_Momentum':
            self.v = 0.9 * self.v + self.lr * grad
            return w - self.v
        elif name == 'Adagrad':
            self.v += grad**2
            return w - (self.lr / (np.sqrt(self.v) + 1e-8)) * grad
        elif name == 'RMSprop':
            self.v = 0.9 * self.v + 0.1 * (grad**2)
            return w - (self.lr / (np.sqrt(self.v) + 1e-8)) * grad
        elif name == 'Adam':
            self.m = 0.9 * self.m + (1 - 0.9) * grad
            self.v = 0.999 * self.v + (1 - 0.999) * (grad**2)
            m_hat = self.m / (1 - 0.9**self.t)
            v_hat = self.v / (1 - 0.999**self.t)
            return w - (self.lr * m_hat) / (np.sqrt(v_hat) + 1e-8)
```

```
# 3. Experiment Runner
def run_experiment(func, grad_func, start_pos, lrs, opt_names, title):
    results = {}
    fig, axes = plt.subplots(1, len(lrs), figsize=(18, 5))
    fig.suptitle(f"Convergence Behavior: {title}", fontsize=16)

    for i, lr in enumerate(lrs):
        for name in opt_names:
            opt = ScratchOptimizers(lr)
            w = np.array(start_pos, dtype=float)
            history = [func(w)]

            start_time = time.time()
            for _ in range(1500):
                g = grad_func(w)
                g = np.clip(g, -10, 10)
                w_new = opt.apply(name, w, g)
                if np.linalg.norm(w_new - w) < 1e-6: break
                w = w_new
                history.append(func(w))
            end_time = time.time()

            axes[i].plot(history, label=name)
            results[(lr, name)] = (w, history[-1], end_time - start_time)

        axes[i].set_title(f"LR = {lr}")
        axes[i].set_yscale('log')
        axes[i].set_xlabel("Iterations")
        axes[i].legend()

    plt.tight_layout(rect=[0, 0.03, 1, 0.95])
    plt.show()
    return results
```

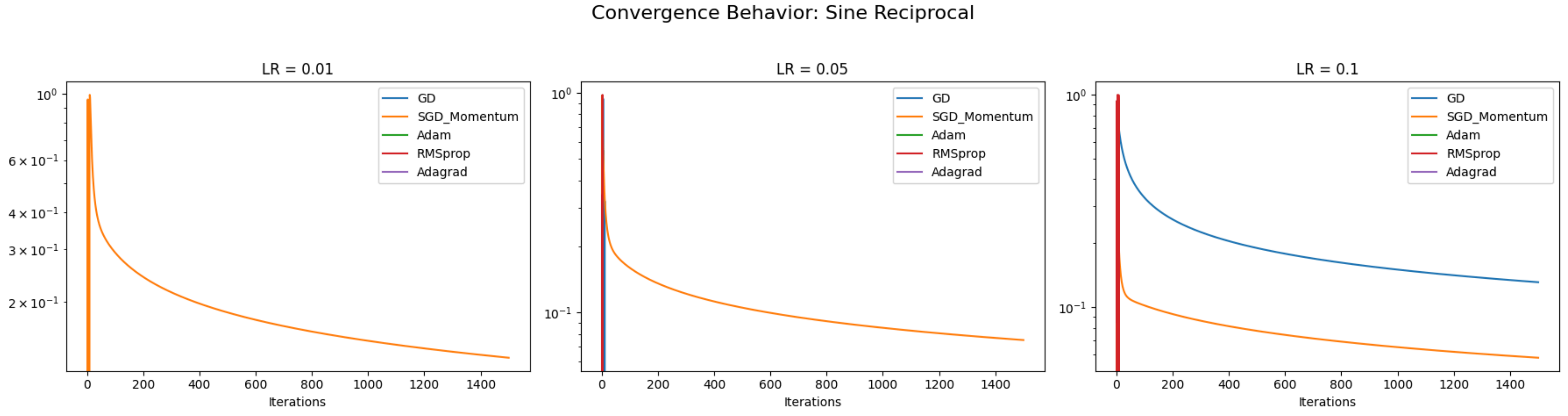
```
# 4. Main Execution
learning_rates = [0.01, 0.05, 0.1]
optimizers = ['GD', 'SGD_Momentum', 'Adam', 'RMSprop', 'Adagrad']

print("Running Rosenbrock Optimization...")
rosen_res = run_experiment(rosenbrock, grad_rosenbrock, [-1.2, 1.0], learning_rates, optimizers, "Rosenbrock Function")

print("\nRunning Sine Reciprocal Optimization...")
# Starting at x=0.2 to see if it moves towards the complex origin
```

```
sine_res = run_experiment(sine_reciprocal, grad_sine_reciprocal, [0.2], learning_rates, optimizers, "Sine Reciprocal")
```

```
Running Rosenbrock Optimization...
```



```
Running Sine Reciprocal Optimization...
```


Convergence Behavior: Sine Reciprocal
LR = 0.01
LR = 0.05
LR = 0.1
GD
SGD_Momentum
Adam
RMSprop
Adagrad
Iterations